Name _____

Computer Architecture
EE 4720
Midterm Examination

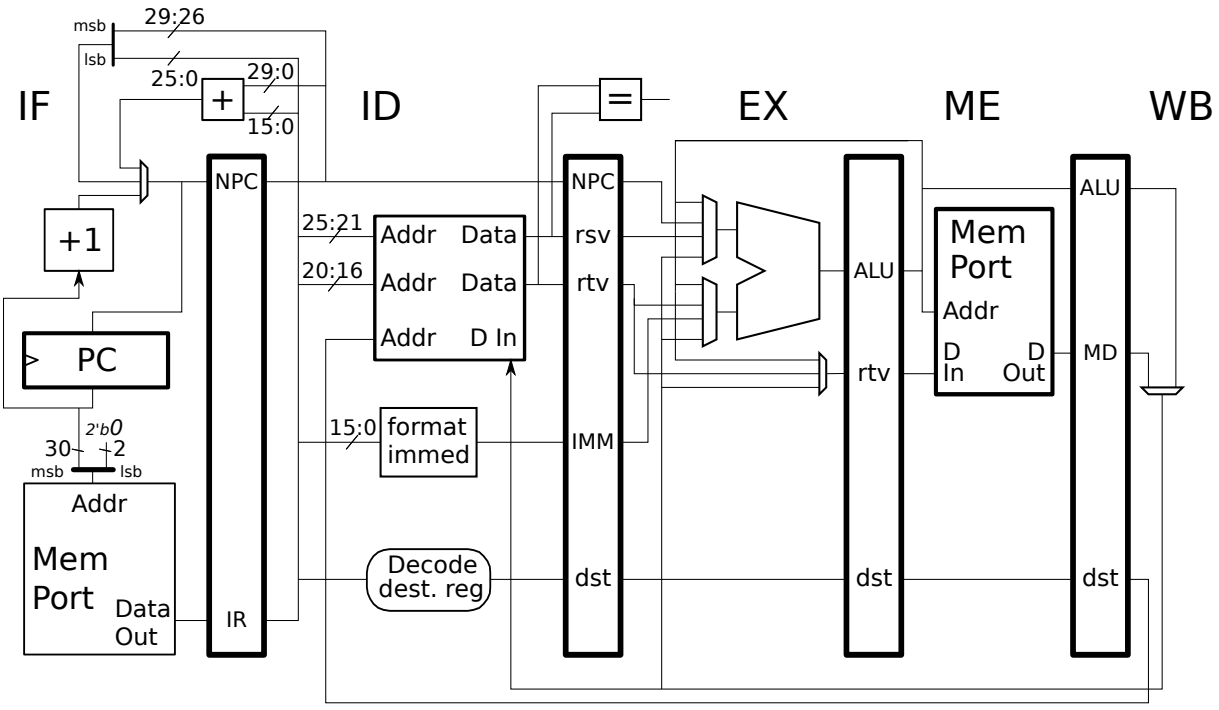
Wednesday, 22 March 2017, 9:30–10:20 CDT

Problem 1	_____	(20 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(20 pts)
Problem 5	_____	(20 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [20 pts] Appearing below is our familiar MIPS implementation.



(*a*) Show pipeline execution diagrams for the code fragments below executing on the illustrated implementation and label as indicated.

☐ Show pipeline diagram. ☐ Doublecheck dependencies.

☐ Label bypass paths used **at mux inputs** with $C : I$, where C is the cycle number (such as 2) and I is the instruction **consuming** the bypassed value. Be sure to check for dependencies.

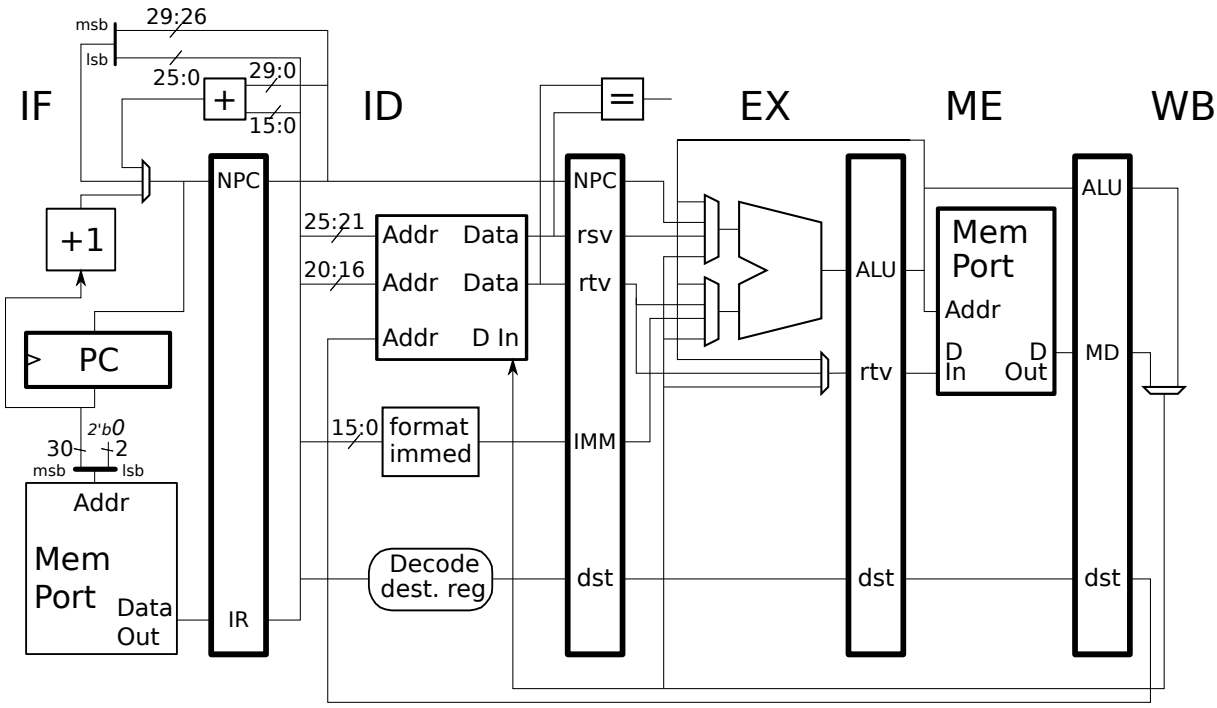
```
add r1, r2, r3

sub r4, r5, r6

sw r4, 8(r1)
```

☐ Show pipeline diagram. ☐ Doublecheck dependencies.

☐ Label bypass paths used **at mux inputs** with $C : I$, where C is the cycle number (such as 2) and I is the instruction **consuming** the bypassed value. Be sure to check for dependencies.

```
and r1, r2, r3

lw r4, 16(r1)

ori r5, r4, 7
```

Problem 1, continued:



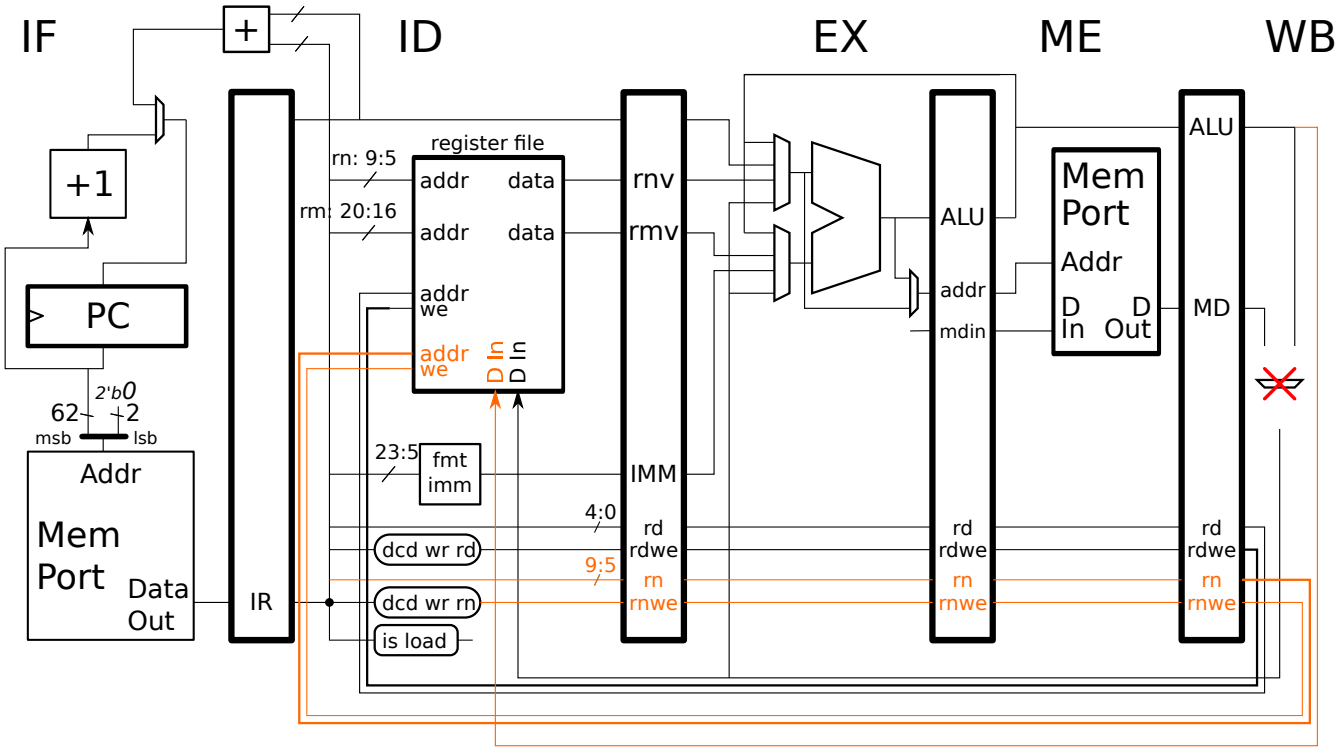
(*b*) Show a pipeline execution diagram for an execution of the code fragment below when the branch is taken. Label the `ID`-stage unit as indicated.

☐ Show pipeline diagram. ☐ Pay close attention to the branch.

☐ Label the **inputs and outputs** of the `ID`-stage unit that computes the branch target. Label with $c : v$, where c is the cycle number and v is the value on the input or output.

```
0x4000: addi r1, r2, 3

0x4004: bne r1, r6, TARG

0x4008: lw r4, 0(r5)

0x400c: addi r5, r5, 4

0x4010: xor r8, r9, r10

TARG:
0x4014: add r5, r4, r4
```

Problem 2: [20 pts] Appearing below is a partial implementation of ARM A64 taken from the solution to Homework 4. The WB-stage mux is crossed out because it's wasteful to use a 64-bit mux when the same functionality can be realized using less expensive logic in the ID stage. For reference, some A64 instructions are shown below, the comments show which field registers are encoded in.




```
@   rd   rn         : Writes rd and rn
ldr x1, [x2], #8    @ x1 = Mem[x2];  x2 = x2 + 8

@   rd, rn, rm      : Writes rd
add x3, x4, x1      @ x3 = x4 + x1

@   rd, rn          : Writes rd
and x1, x2, #34     @ x1 = x2 & 34;
```

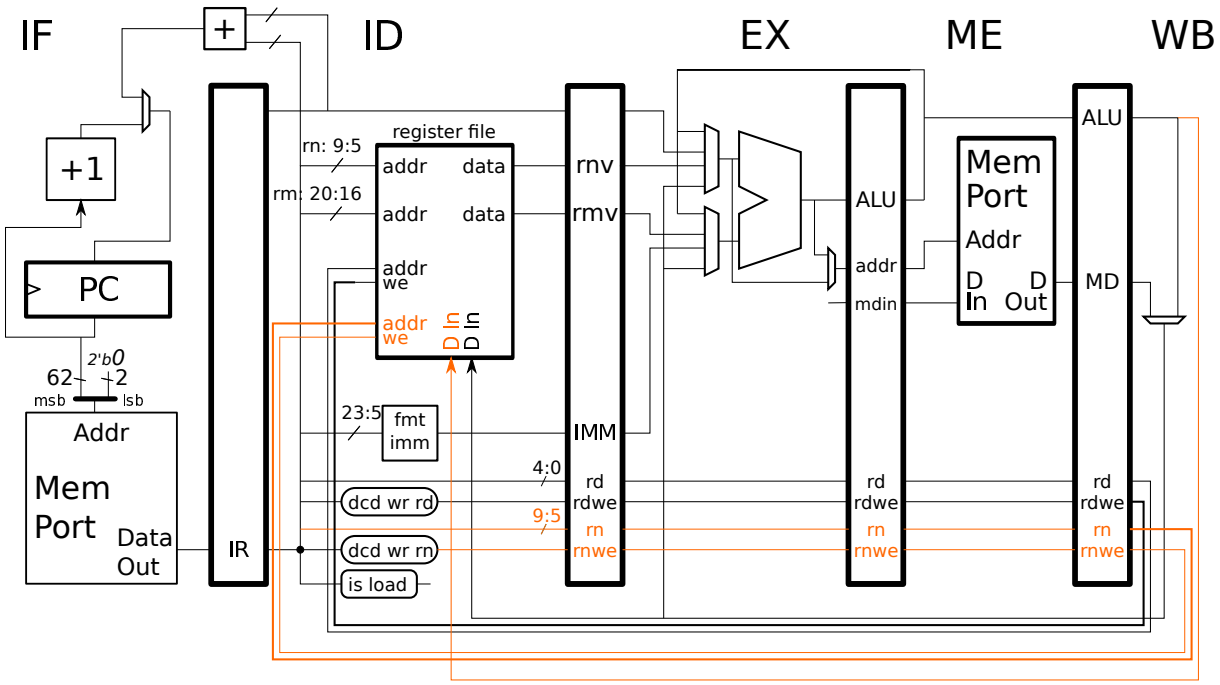
Complete the changes so that instructions such as the ones above can write back their results. Assume that only the load instructions write the rn-field register.

☐ In WB consider re-connecting wires broken by the removal of the mux.

☐ In ID make changes so that instruction results can be written back to the correct registers.

☐ Make the best use of the existing from-WB bypass path.

Problem 3: [20 pts] Appearing below again is the partial implementation of ARM A64 taken from the solution to Homework 4.




```
@   rd   rn         : Writes rd and rn.  Post-index
ldr x1, [x2], #8    @ x1 = Mem[x2];  x2 = x2 + 8

@   rd   rn         : Writes rd and rn.  Pre-index
ldr x1, [x2, #8]!   @ x2 = x2 + 8;  x1 = Mem[x2];

@   rd   rn  rm
str x1, [x2, x3]    @ Mem[x2+x3] = x1
```

(*a*) Make changes needed to implement the store instruction, see the example above. Just show datapath, not control logic.

☐ Changes for the store instruction.

(*b*) Do we really need *both* pre-index and post-index addressing for loads and stores? Eliminating either of them will reduce cost, but one's elimination would reduce cost by more than the other's. Indicate which saves more and show the hardware that can be removed and other needed changes. *Note: The phrase* and other needed changes *was not in the original exam.*

☐ Greater cost reduction by eliminating: ◯ *pre-index* ◯ *post-index* (check exactly one).

☐ Show the hardware that's not needed ☐ and other needed changes.

Problem 4: [20 pts] When MIPS routine `coursei` is called register `a0` will hold an entry number, referring to the table at label `courses`. Complete the routine so that when it returns register `v0` will have the integer representation of entry number `a0` in the table. Note that the table itself holds floats. For example, when called with `a0=0` it should return with `v0=2740`, when called with `a0=2` it should return with `v0=3755`, etc.

☐ Complete so `v0` is integer representation of `a0`th table entry.

☐ Read the table as it is, don't modify it or read a different table.

```
        .data
courses:
        .float 2740
        .float 3750
        .float 3755
        .float 4755
        .float 4720
        .float 7722
        .float 7725
        .text
        # CALL VALUE: $a0: Entry in table to look up.
        # RETURN:     $v0: Table entry #$a0 represented as an integer.
coursei:
        la $t0, courses









        jr $ra
        nop
```

Problem 5: [20 pts] Answer each question below.

(*a*) What kind of implementations were RISC ISAs designed to simplify?

☐ Kinds of implementations that RISC designed to simplify:

(*b*) Describe how the features below simplify RISC ISA implementations.

☐ Fixed-size instructions.

☐ Avoiding arithmetic instructions that access memory.

(*c*) The SPECcpu package contains the source code for the SPEC benchmarks and scripts to compile and run them, but it does not come with compilers. The tester provides his or her own. Consider a *SPECcpu+* package that comes with compilers, and the requirement that those compilers be used. Why would that make SPECcpu+ less useful to computer engineers?

☐ SPECcpu+ less useful because:

(*d*) In class we described some optimizations as high-level, and some as low-level, performed by the back end. What distinguishes high- and low-level optimizations? Provide an example of a low-level optimization that could only be performed by the compiler back end.

☐ Difference between high- and low-level optimization.

☐ Example of an optimization that must be low-level (that can only be done in the back end).

☐ Briefly explain why.

(*e*) MIPS I has instruction `bgtz r1, TARG` in which the branch is taken if `r1>0` but it lacks an instruction like `bgt r1, r2, TARG` that would branch if `r1>r2`. Why?

☐ Why does MIPS lack `bgt r1, r2, TARG`?

☐ What would be the impact on performance of including `bgt r1, r2, TARG` in MIPS on "our" five-stage implementation?